VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra telekomunikační techniky

Komunikační server s vysokou dostupností

High availability communication server

2009/2010 **Libor Trávníček**

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7.5.2010 podpis……..

Abstrakt

Cílem této práce je najít jednotlivá „open-source” řešení a poté pomocí některých z nich navrhnout a implementovat SIP server s vysokou dostupností. Mezi problémy, které se musí u tohoto vysoce dostupného serveru vyřešit se řadí výměna IP adresy mezi servery, a přenos dat nutných k správné činnosti. Tyto data, v případě SIP serveru registrační údaje, se musí tzv. replikovat automaticky a v pravidelných intervalech.

Klíčová slova

Databáze,open-source, replikace, telekomunikace, vysoká dostupnost

Abstract

The aim of this thesis is to find a single "open-source" solutions and then with several of them propound and implement high availability SIP server. Among problems, which have to be solved at high availability server are changeover of the IP address between servers and transfer of the data necessary for the correct function. These data, in a case of SIP server registration data, have to be so-called replicated automatically and within regular interval.

Key words

Database, high availability, open-source, replication, telecommunication

Seznam použitých zkratek:

ARP	Address Resolution Protocol
B2BUA	Back-To-Back User Agent
CARP	Common Address Redundancy Protocol
DRBD	Distributed Replicated Block Device
DSN	Database Source Name
GPL	GNU Public Licence
HSRP	Hot Standby Router Protocol
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISO/OSI	International Organization for Standartization / Open Systems Interconnections
LAN	Local Area Network
LVS	Linux Virtual Server
MAC	Media Access Control
ODBC	Open DataBase Connectivity
SIP	Session Initiation Protocol
SSH	Secure Shell
STONITH	Shoot The Other Node In The Head
TCP	Transmission Control Protocol
VoIP	Voice over Internet Protocol
VRID	Virtual Router Identifier
VRRP	Virtual Router Redundancy Protocol

Obsah

1 Úvod

Novou technologii VoIP nahrazující stávající telefonní síť je stále možné zdokonalovat. Jednou z možností zdokonalení je zajištění vyšší dostupnosti SIP serveru, hlavně pak registračního, jenž je důležitým bodem v architektuře. Na registrační server se můžou registrovat uživatelé telefonů, aby pak již mohli využívat další klasické telefonní služby, jako například volání na ostatní telefonní účastníky.

Mým úkolem je zvýšit dostupnost asterisk SIP serveru, který funguje na principu B2BUA (Back-To-Back User Agent). Uskutečněné volání tedy neprobíhá přímo mezi volajícím a volaným, ale asterisk se jeví pro volajícího jako volaný a pro volaného jako volající.[9]

V první kapitole s názvem **„**pohled na IP telefonii z pohledu spolehlivosti**”** se zabývám typy poruch, které můžou nastat v případě SIP serveru a jak se dají řešit vnitřní poruchy SIP serveru metodami vysoké dostupnosti.

Ve druhé kapitole pak rozebírám jednotlivé metody, pomocí kterých se dá zajistit vysoká dostupnost a jejich možnou použitelnost v případě SIP serveru asterisk.

Ve třetí kapitole jsem pak vybral nejlepší řešení a je zde popsána jejich praktická implementace na 2 stanice tak, aby byl vytvořen vysoce dostupný SIP server.

V poslední čtvrté kapitole pak pomocí síťového analyzátoru a praktických poznatků analyzuji funkčnost celého systému.

2 Pohled na IP telefonii z pohledu spolehlivosti

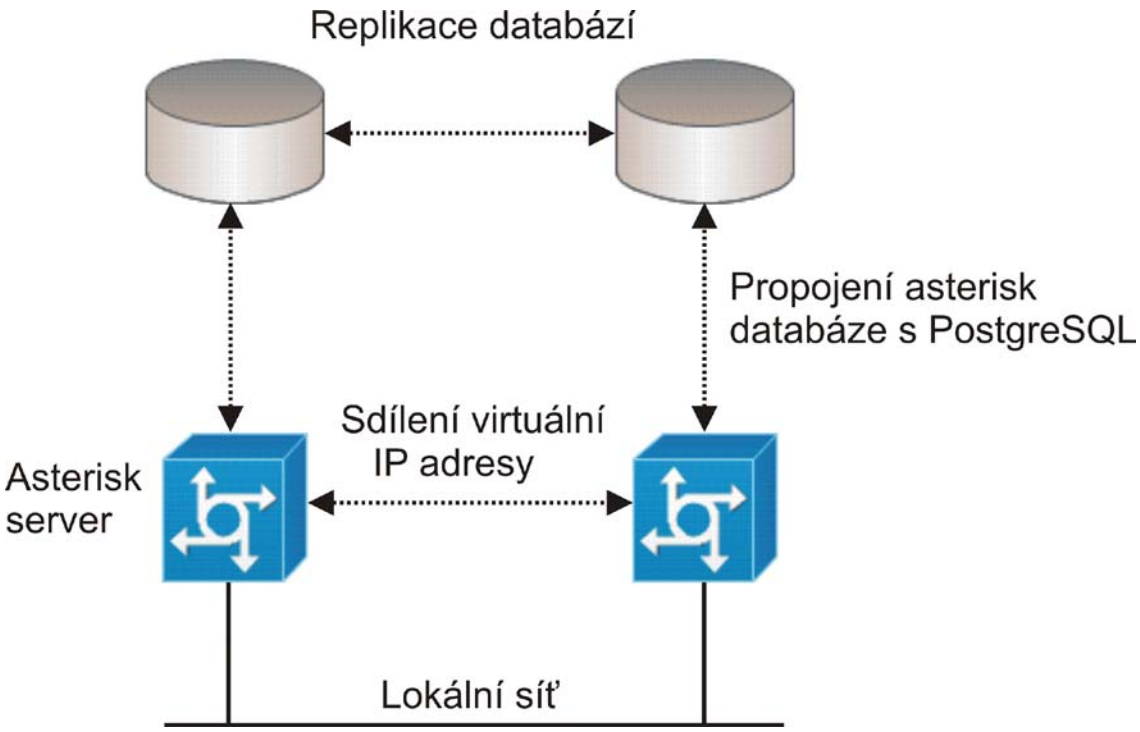
V této době se stále více uplatňují nové technologie pro přenos hlasu přes IP sítě nazývané obecně jako VoIP (Voice over IP), jde hlavně o protokol SIP (Session Initiation Protocol). Pro jejich další rozšiřování, hlavně při použití v komerčním sektoru je třeba zajistit určitou spolehlivost služby VoIP, jako pravděpodobný nástupce standartní telefonní sítě.

Mezi hlavní problémy, které můžou nastat v architektuře používající protokol SIP patří:

- zahlcení SIP serveru
- porucha SIP serveru nebo některé z komponent LAN sítě (switch, kabely)
- výpadek energie z napájecí sítě
- softwarová chyba v SIP serveru

V této práci bych se chtěl hlavně zabývat druhým problémem, je více pravděpodobným, dá se omezit různými řešeními, na druhou stranu jsou tyto řešení více složitější, než zvýšit např. propustnost systému. Jednotlivé použité prvky systému budou "open source".

Hlavní podstatou omezení problému při poruše SIP serveru je mít zapojen záložní identický server, který může převzít činnost za porouchaný v co nejrychlejším čase, pravděpodobně v řádu jednotek sekund, tedy v režimu "hot standby", v češtině by se dal tento termín přeložit jako horká záloha. Pro potřeby SIP je navíc potřeba, aby náhradní server převzal IP adresu předchozího serveru pro správnost směrování a navíc je potřeba mít k dispozici na záložním serveru databázi registrací, identickou s databází nacházející se na hlavním serveru. Tím pádem pak uživatelé zaregistrovaní v minulosti na hlavním server budou i po přechodu na záložní server nadále zaregistrováni.



Obr. 1: Zamýšlené zapojení komunikačních serverů s vysokou dostupností.

3 Nástroje pro zvýšení dostupnosti a redundance VoIP systémů

Na úvod bych poznamenal, že jako SIP server bude sloužit Asterisk, pro databázovou službu použiji PostgreSQL databázi.

Nejdříve bych zde nastínil několik možností řešení týkajících se implementace tzv. horké zálohy a zároveň výměny IP adresy mezi servery. Mezi nejnadějnější adepty patří VRRP (Virtual Router Redundancy Protocol), Keepalived projekt a Heartbeat, oba pod GPL licencí.

Dále je potřeba vytvořit spojení SIP serveru s databázovým serverem, který poté bude zajišťovat potřebné kapacity pro správný chod SIP registrací namísto vlastních databází SIP serveru. K tomuto účelu existuje tzv. ODBC (Open DataBase Connectivity) konektor, schopný propojit aplikace kompatibilní s ODBC na množství různých databázových systémů.

V poslední části bych představil možné řešení replikace dat mezi databázovými servery. Replikace se může rozlišovat v několika aspektech. Prvním aspektem je vlastnost, jak se bude provádět replikace mezi jednotlivými servery.

- master / slave – master server je schopen ukládat data a číst z nich, slave pouze kopíruje data z master serveru, případně se z něj dají data číst.

- master / master – oba servery jsou schopny ukládat a číst data, posílaji si mezi sebou změny. Problém může nastat při společném zápisu podobných dat na obou serverech.

- multi master – Stejný případ jako master / master pro počet serverů větší než 2.

Dalším rozlišením je způsob provádění kontroly zápisu.

- synchronní – Při zápisu dat má úspěšný zápis platnost až po potvrzení zápisu na všech serverech. Nevýhodou je větší časová prodleva.
- asynchronní – Na potvrzení od ostatních serverů se nečeká, může se projevit nevýhoda v případě výpadku hlavního serveru nekonzistentností dat.

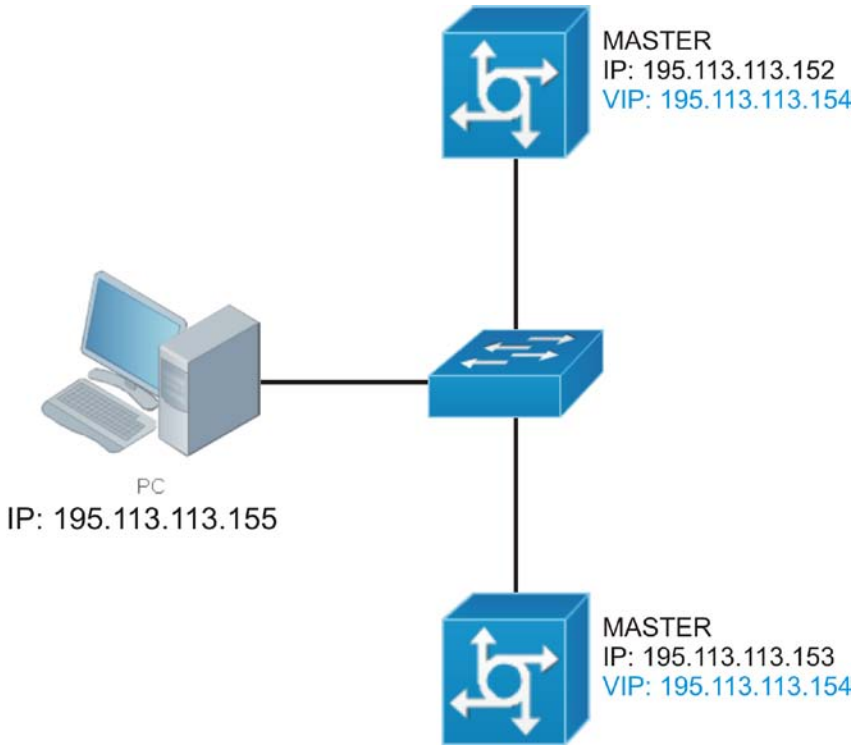
První možností je využití DRBD (Distributed Replicated Block Device), což je aplikace kopírující uložiště dat z aktivního serveru na záložní. V našem návrhu bude aktivní vždy jen jeden server s virtuální IP adresou, na kterou se můžou SIP uživatelé registrovat, Po přechodu virtuální IP adresy se stane aktivním druhý server a ODBC bude kopírovat opačným směrem. Problém by nastal, pokud by se někdo registroval na reálnou IP adresu neaktivního serveru.

Pro nás je nejvíce užitečným řešením použití replikace pouze dat z databáze. Vzhledem k celkové architektuře vysoce dostupného systému je třeba se zaměřit na replikace typu master-master, poněvadž u replikace master-slave slouží záložní databáze pouze ke čtení dat. Jako nejvhodnější se jeví pro databázový systém PostgreSQL synchronní multi-master systém PGCluster. Mezi další použitelné se řadí Bucardo, Slony-1 nebo Sequoia.

3.1 Nástroje pro vytvoření sdílení virtuální IP adresy

3.1.1 VRRP

Základním prvkem pro funkci sdílení virtuální IP adresy dvěmi stroji na základě jejich okamžitého stavu je VRRP pracující na úrovni spojové a sí'tové vrstvy modelu ISO/OSI. Byl primárně určen pro nahrazení jednoho routeru druhým, pokud první vypoví službu.



Obr. 2: Znázorněná funkce protokolu VRRP

Na dvou stanicích, na kterých je nainstalován VRRP daemon se nastaví virtuální IP adresa, kterou chceme sdílet těmito stanicemi, je možné nastavit i více adres (obr. 2). Je možné použít reálnou adresu jedné ze stanic, v našem případě je však spolehlivější použít novou adresu z rozsahu adres

naší sítě.Je třeba nastavit VRID (Virtual Router Identifier), podle kterého se dají odlišit obě stanice. Pak je také možné nastavit, která bude hlavní stanicí a která záložní. To se děje nastavením priority od 0 do 255, kdy 0 znamená nefunkční a 255 je nejvyšší priorita.

Po inicializaci VRRPd na stanicích se rozhoduje o hlavní stanici podle kritérií v pořadí – Stanice má identickou IP adresu s virtuální IP adresou, pokud ne, pak se rozhoduje podle priority a pokud je i priorita totožná, pak se vybírá stanice s vyšší numerickou hodnotou IP adresy.

<table>
<tr><td colspan="32">0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td></tr>
<tr><td colspan="8">Version</td><td colspan="8">Type</td><td colspan="8">Virtual Router ID</td><td colspan="8">Priority</td><td colspan="8">Count IP Addresses</td></tr>
<tr><td colspan="16">Auth Type</td><td colspan="8">Advertisement Int</td><td colspan="8">Checksum</td></tr>
<tr><td colspan="32">IP Address (1)</td></tr>
<tr><td colspan="32">.
.
.</td></tr>
<tr><td colspan="32">IP Address (n)</td></tr>
<tr><td colspan="32">Authentication Data (1)</td></tr>
<tr><td colspan="32">Authentication Data (2)</td></tr>
</table>

Tabulka č.1: VRRP paket

Ta pak posílá v pravidelných intervalech, standartně nastaveno na 1s, VRRP pakety (Tabulka č.1) na speciální VRRP multicast 224.0.0.18 s informacemi o virtuálních IP adresách a další nezbytné údaje v hlavičce. Standartně je v hlavičce nastavená priorita pro master router 255 a pro záložní 100, dále se zde objevuje verze protokolu, typ paketu, jediný definovaný je typ Advertisement. Je uveden VRID, typ autentizace (žádná, textové heslo nebo IP autentizační hlavička), interval ohlašování. Pokud po uplynutí intervalu ohlašování master nepošle VRRP paket nebo ve VRRP paketu nastaví svou prioritu na 0, stane se náhradní stanice masterem a převezme virtuální IP adresu.[1]

3.1.2 Keepalived

Keepalived vytvořený skupinou Linux Virtual Server má za cíl poskytovat dvě hlavní funkce:

- Rozkládání zátěže na více strojů.
- Sdílení virtuální IP adresy mezi dvěmi stroji.

V keepalived je implementován VRRP vysvětlený dříve a jsou k němu přidány další funkce.

Vylepšen je přechod virtuální IP adresy, kdy stanice, která přejde do stavu master pošle na switch ethernet rámec ARP Gratuitous, díky kterému si switch upraví podle informací v rámci ARP tabulku – změna MAC adresy uvedené IP adresy. Tím je urychleno posílání paketů na nový master.[2]

Je zde možnost provádět tzv. Healtcheck, tedy nastavit v konfiguraci Keepalived pravidelnou kontrolu aktivit TCP, HTTP provozu druhé stanice, či případně spouštění vlastního skriptu. Dále je možné nastavit v konfiguraci skripty, které se budou spoušťet v případě vybrané změny stavu – při přechodu na master, backup nebo do chybného stavu. [3]

3.1.3 Heartbeat

Hodně rozšířeným daemonem pro funkci vysoce dostupného serveru je Heartbeat vytvořený projektem High Availability Linux, nyní již ve verzi 2.1.4. Funkcí je hodně podobný výše zmíněnému projektu Keepalived. Zajišťuje základní funkci výměny IP adresy, spouštění služeb podle nastavení v konfiguraci, od verze 2 podporuje i spojení více než dvou počítačů do clusteru.

Pomocí Heartbeat dokáže záložní server rozpoznat, zda-li je hlavní server v pořádku a běží na něm požadované služby. Pokud ne, převezme za něho IP adresu a spustí stejné služby na svém hardwaru. Správnou funkci celého clusteru kontroluje tzv. STONITH (Shoot The Other Node In The Head) mechanismus, který zajišťuje, aby např. porouchaný server nezačal posílat nekorektní data, či jiné nechtěné operace. Pokud se tak stane, může server restartovat.

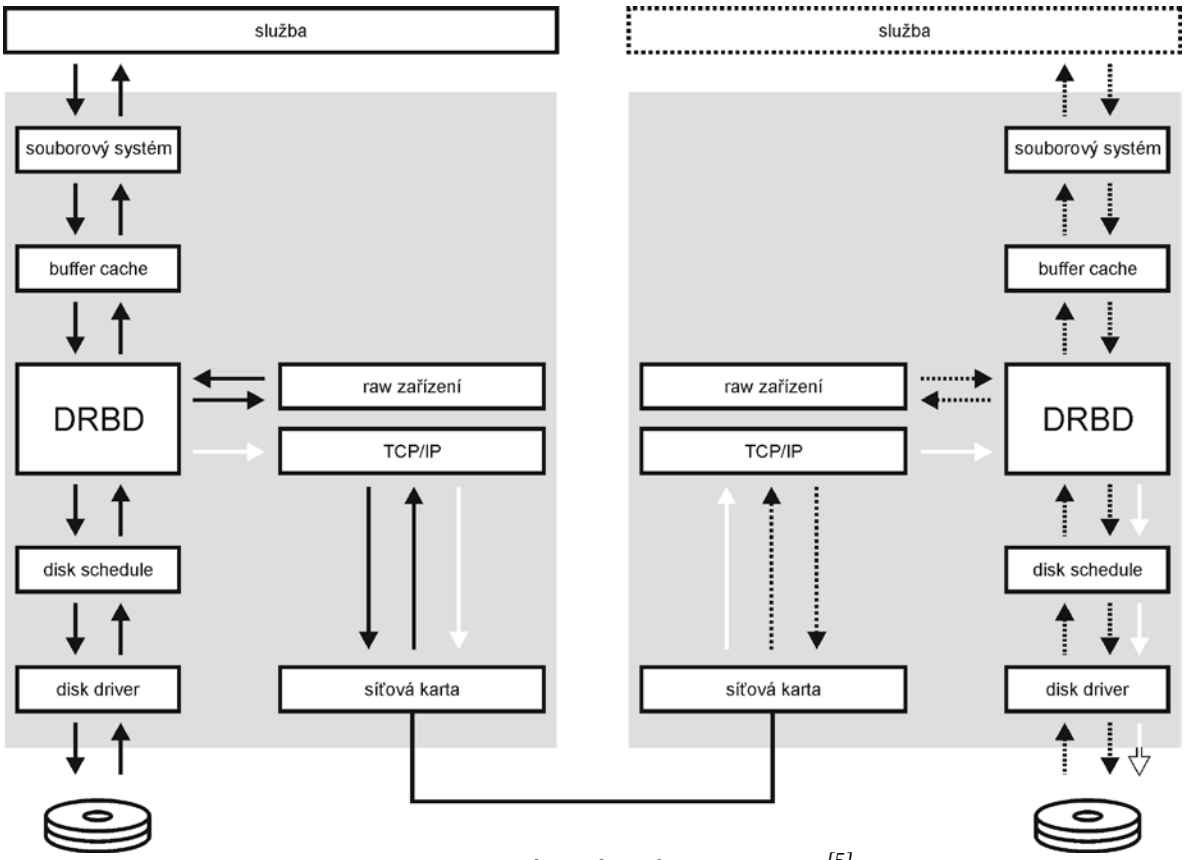
Heartbeat může dále spolupracovat s programy zajišťujícími rozložení zátěže na několik serverů, např. LVS (Linux Virtual Server), nebo ClusterIP. Dále také podporuje zrcadlení souborových systémů DRBD (Distributed Replicated Block Device). [4]

Mezi další možnosti patří protokol HSRP (Hot Standby Routing Protocol) od firmy Cisco z kterého se standardizací vytvořil protokol VRRP nebo protokol CARP (Common Address Redundancy Protocol).

3.2 Databázová replikace

3.2.1 DRBD

Řešením replikace databází může být DRBD (Distributed Replicated Block Device), které kopíruje zápisy na datové uložiště z jednoho serveru na druhý synchronně nebo asynchronně podle vlastní volby, nevýhodou je omezení pouze na dvě uložiště, což nás ovšem v našem případě mnoho neomezuje. Druhou nevýhodou je potřeba propojení s velmi vysokou přenosovou rychlostí. Vždy uložiště jednoho serveru (zrovna aktivního) by bylo master a druhé by bylo v režimu slave. DRBD operuje mezi souborovým systémem a samotným diskem (obr. 3).



Obr. 3: Blokové schéma funkce DRBD[5]

3.2.2 PGCluster

PGCluster je synchronní multi-master replikační systém postavený na databázovém serveru PostgreSQL. Skládá se ze tří funkčních bloků:

- cluster DB – samotná databáze, používá patch pro replikaci na databázový systém PostgreSQL, cluster DB po přijetí dotazu začne vykonávat svou činnost, zároveň přeposílá dotaz replikačnímu serveru.

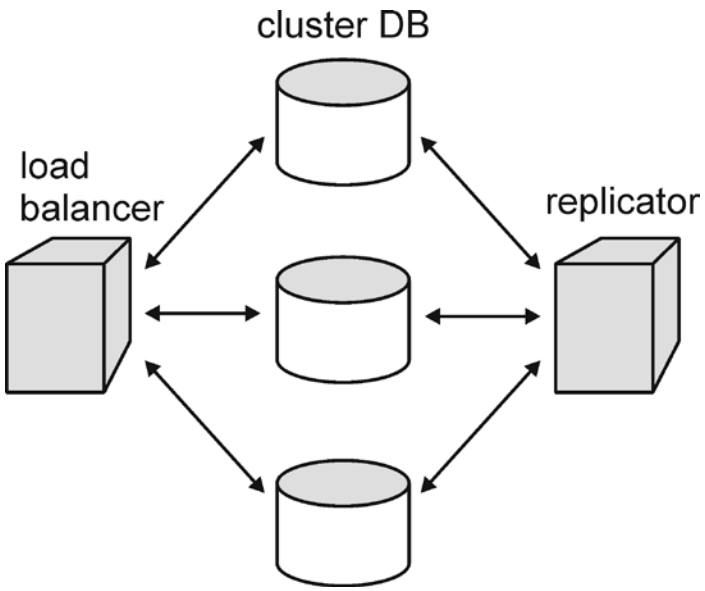
- replicator – replikační server přeposílá dotazy z jedné databáze na všechny, dotazy přeposílá v pořadí v jakém přišly. Kontroluje taky stav cluster databází,

pokud zaregistruje problém, odpojí problémovou cluster DB z replikační funkce. Po zapojení cluster DB jí synchronizuje data.

- load balancer – blok zajišťující rozložení zátěže, po přijetí dotazu od klienta přeposílá dotaz na nejméně zatíženou cluster DB. Load balancer taktéž jako replicator kontroluje stav jednotlivých cluster DB.

S pomocí těchto bloků můžeme vytvořit dvě hlavní funkce:

- Rozložení zátěže
- Vysoká dostupnost

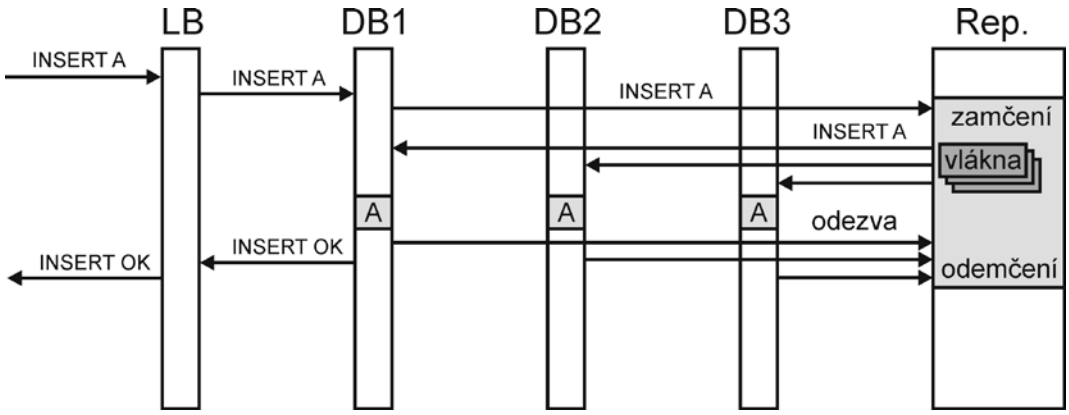


Obr.4: Celková kompozice PGCluster

PGCluster replicator replikuje pouze zápisy do databáze na ostatní databáze, tím se snižuje potřebné přenosové pásmo. Při obnově nebo přidání nového databázového serveru se používá program rsync, který kopíruje adresář obsahující data databáze z již funkčního serveru na nový server, je schopen

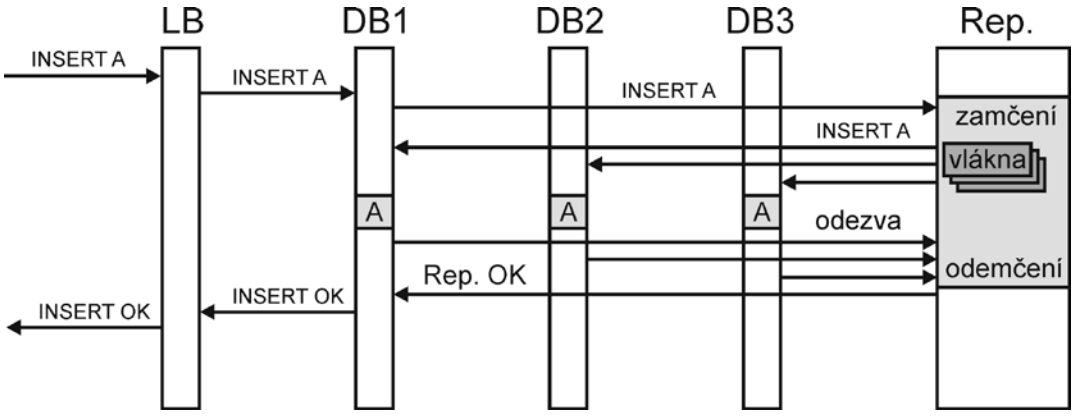
však vynechat kopírování již existujících dat a kopírovat jen data nevyskytující se na novém serveru.

U replikace je možné využívat dva režimy. Prvním režimem je normal mode, kdy se posílá potvrzení zápisu po úspěšném zápisu v databázovém serveru, který byl dotazován.



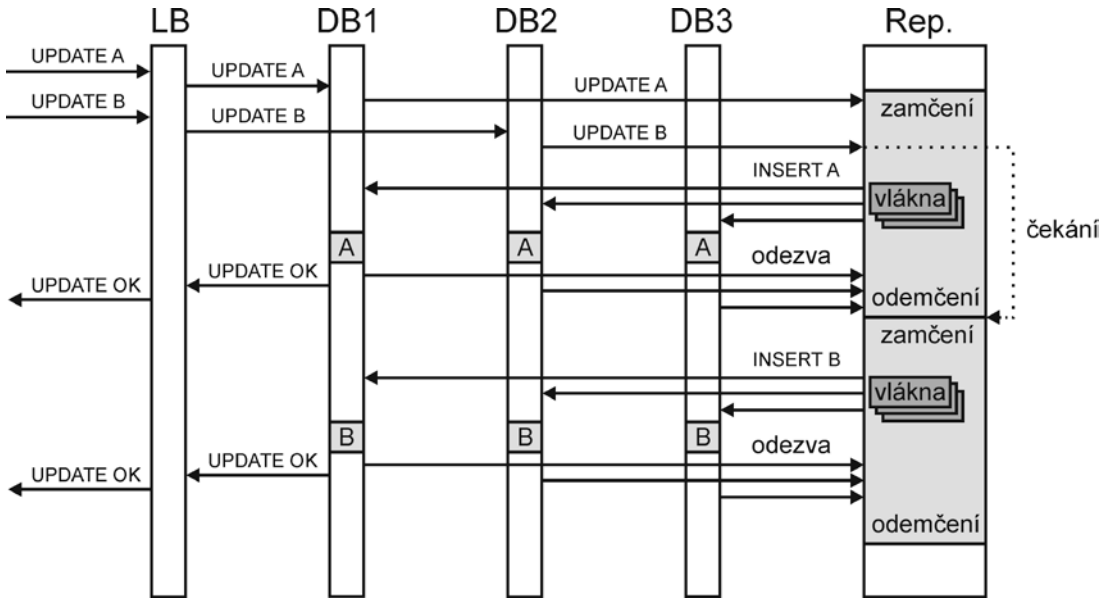
Obr.5: normal mode

Druhým režimem je reliable mode, kdy se čeká s posláním potvrzení zápisu až po obdržení potvrzení od replikačního serveru o úspěšném zápisu na všechny databáze.



Obr. 6: Reliable mode

Po obdržení požadavku o zápis se nejdříve replikační server zamkne a nepřijímá další požadavky. Poté vytvoří pro každý databázový server samotné vlákno, starající se o zápis na svůj server. Po obdržení kladné odezvy skončí vlákno svou činnost. Replikační server se pak odemkne po ukončení všech vláken.



Obr. 7: Příklad funkce při více souběžných transakcích

Při výpadku jedné z cluster DB a opětovném spuštění je nutné zahájit tzv. recovery (obnovu) dat, kdy se přes replikační server nové data, které se v průběhu výpadku nemohly replikovat, obnovují z druhé cluster DB.[6]

4 Implementace řešení vysoce dostupného komunikačního serveru

Na obou stanicích, virtualizovaných pomocí VMserveru je nainstalována linuxová distribuce debian verze 5.0.4 a asterisk 1.6.1.18. Jako databázový a zároveň replikační server jsem použil aplikaci PGCluster v1.9.0rc5 postavenou na databázovém serveru PostgreSQL v8.3.

Obvykle se pro potřeby SIP uživatelů ústředny asterisk v konfiguračním souboru *sip.conf* vytvoří uživatel či uživatelé následujícím způsobem:

```
[1200]              # telefonní číslo
type=friend         # typ uživatele
context=phones      # zařazení v určité směrovací skupině
host=dynamic        # určení pevné nebo dynamické IP adresy
```

Po uložení a restartu asterisku se již může uživatel s telefonním číslem 1200 zaregistrovat na ústředně, pokud zná IP adresu ústředny. Po úspěšné registraci uživatelova telefonu si asterisk ukládá některé další nutné údaje o telefonu:

- IP adresa
- port
- počet sekund do reregistrace telefonu

Tento jednoduchý příklad se zkomplikuje v případě nutnosti zajistit vyšší dostupnost ústředny použitím dvou identických serverů, které se v případě poruchy jednoho mohou zastoupit.

Pro vytvoření vysoce dostupné ústředny asterisk je třeba zajistit 2 hlavní podmínky:

- Záložní server si musí v případě výpadku hlavního serveru převzít jeho IP adresu, tak aby z pohledu uživatele byl server stále na stejné IP adrese.

- Záložní server také potřebuje k činnosti údaje o aktuálních registracích všech telefonů pokud chceme, aby se nemusel uživatel po výpadku znovu registrovat.

4.1 Zajištění přenosu IP adresy

Prvním krokem je tedy zajistit přenositelnost IP adresy mezi dvěma servery. K tomuto účelu jsem si vybral program keepalived v1.1.15.

Jako základ je použit VRRP (Virtual Router Redundancy Protocol) jehož hlavní funkcí je právě rozhodování o tom, která ze stanic bude právě využívat vybranou IP adresu (tzv. Virtuální IP adresa) podle nastavené priority stanic. Server vybraný na začátku jako hlavní díky vyšší nastavené prioritě v konfiguraci posílá v pravidelných intervalech paket určený pro záložní server na vrrp multicast adresu 224.0.0.18 kde popisuje svou prioritu. Pokud nastaví svou prioritu na 0 nebo záložní server neobdrží v určeném intervalu paket, záložní server si zvýší prioritu a stane se držitelem IP adresy. Pokud se po čase opět spustí hlavní server, IP adresu si opět převezme.

Použité stanice mají IP adresy 195.113.113.152 a 195.113.113.153, virtuální IP adresu jsem vybral 195.113.113.154. V konfigurace keepalived pak nastavím na každé stanici vybrané údaje:

- vrrp_instance – název bloku konfigurace VRRP instance.
- state - stav stanice v normálním provozu MASTER nebo BACKUP.
- interface – na kterém síťovém zařízení se má pracovat.
- virtual_router_id - identifikátor virtuálního routeru.
- priority – prioritu (větší slovo má však atribut state).
- advert_int - velikost časového intervalu posílání VRRP paketů v sekundách.
- authentication – blok s údaji o autentizaci.
- auth_type – typ, PASS nebo AH (Authentication Header).

- auth_pass – heslo stejné pro všechny stanice ve virtuálním routeru.
- virtual_ipaddress – rozpis virtuálních IP adres.

Na hlavním serveru je konfigurace nastavena takto.

```
# /etc/keepalived/keepalived.conf
vrrp_instance HA1 {                    # nazev vrrp instance
        state MASTER
        interface eth0
        virtual_router_id 51           # id virtualniho serveru
        priority 200
        advert_int 1                   # interval posílání vrrp multicast paketů
        authentication {
                auth_type PASS         # k autentizaci serverů
                auth_pass 9786
        }
        virtual_ipaddress {
                195.113.113.154/27
        }
}
```

Na záložním serveru je konfigurace nastavena takto.

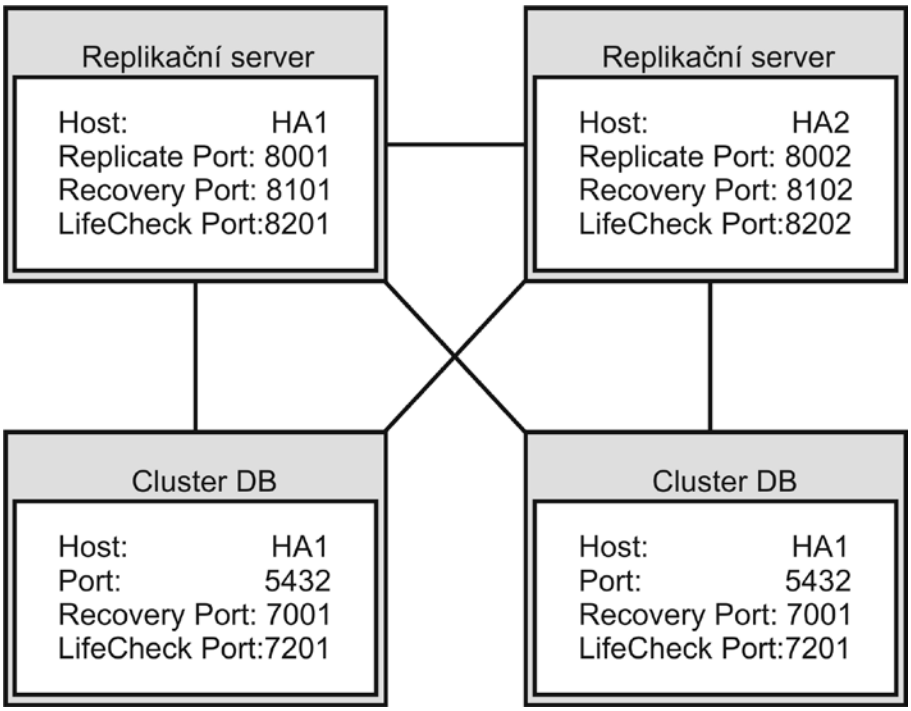
```
# /etc/keepalived/keepalived.conf
Vrrp_instance HA1 {                    # nazev vrrp instance
        state BACKUP
        interface eth0
        virtual_router_id 51           # id virtualniho serveru
        priority 50
        advert_int 1                   # interval posílání vrrp multicast paketů
        authentication {
                auth_type PASS         # k autentizaci serverů
                auth_pass 9786
        }
        virtual_ipaddress {
                195.113.113.154/27
        }
}
```

Po restartu stanic se již můžeme připojit k serveru s IP adresou 195.113.113.154, bez ohledu, zda-li je to ve skutečnosti stanice s adresou 195.113.113.152 nebo 195.113.113.153. Pokud je však funkční stanice s nastavením state MASTER, bude držitelem virtuální IP adresy. Stanice s nastavením BACKUP se stává držitelem jen do té doby, než je opět MASTER funkční.

4.2 Instalace a konfigurace PGCluster

Nejdříve jsem použil PGCluster ve verzi 1.9.0rc6, ale po problémech se spuštěním replikačního serveru jsem přešel na verzi 1.9.0rc5.

Na každém serveru bude nakonfigurována jak cluster DB, tak replikační server. Replikační server je taktéž zálohován, aby byl v případě výpadku jednoho serveru záložní replicator, z kterého je možné poté obnovit cluster DB.



Obr. 8: Schéma propojení komponent PGCluster

4.2.1 Instalace PGCluster

Před samotnou instalací PGCluster je nutné nainstalovat další potřebné programy, zlib, openssl, openssh a rsync.

Nejdříve je nutné vytvořit uživatelský účet se jménem postgres, který bude sloužit pro administraci databázového serveru. Poté zkompilujeme a nainstalujeme pgcluster a provedeme inicializaci databáze.

```
# adduser postgres
# cd /usr/src/pgcluster-1.9.0rc5
# su postgres
$ ./configure
$ make
$ su
# make install
```

PGCluster je nainstalován v adresáři /usr/local/pgsql, vytvoříme v něm novou složku data, kde se budou ukládat data databáze. a poté tomuto adresáři přiřadíme jako vlastníka uživatele postgres. Inicializaci provedeme příkazem initdb s volbou -D, kde uvedeme umístění databázového adresáře.

```
# mkdir /usr/local/pgsql/data
# chown -R postgres /usr/local/pgsql
# su postgres
$ /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
```

4.2.2 Konfigurace PGCluster

Tím máme připraven databázový systém, nyní je třeba nastavit konfiguraci cluster DB v souboru cluster.conf a replikačního serveru v pgreplicate.conf.

```
#/usr/local/pgsql/data/cluster.conf
# udaje o replikacnich serverech a jejich portech.
<Replicate_Server_Info>
        <Host_Name>         HA1         </Host_Name>
        <Port>              8001        </Port>
        <Recovery_Port>     8101        </Recovery_Port>
        <LifeCheck_Port>    8201        </LifeCheck_Port>
</Replicate_Server_Info>
```

```
<Replicate_Server_Info>
        <Host_Name>          HA2              </Host_Name>
        <Port>               8002             </Port>
        <Recovery_Port>      8102             </Recovery_Port>
        <LifeCheck_Port>     8202             <LifeCheck_Port>
</Replicate_Server_Info>
# nastaveni samotne cluster DB
<Recovery_Port>              7001             </Recovery_Port>
<LifeCheck_Port>             7201             </LifeCheck_Port>
<Rsync_Path>                 /usr/bin/rsync   </Rsync_Path>
<Rsync_Option>               ssh              </Rsync_Option>
<Rsync_Compress>             no               </Rsync_Compress>
<Rsync_Timeout>              10min            </Rsync_Timeout>
<Rsync_Bwlimit>              0KB              </Rsync_Bwlimit>
<Pg_Dump_Path>     /usr/local/pgsql/bin/pg_dump          </Pg_Dump_Path>
<Ping_Path>        /bin/ping                             </Ping_Path>
<When_Stand_Alone>           read_write       </When_Stand_Alone>
<Replication_Timeout>        1min             </Replication_Timeout>
<LifeCheck_Timeout>          3s               </LifeCheck_Timeout>
<LifeCheck_Interval>         15s              </LifeCheck_Interval>
```

U cluster DB je nutné nastavit přístupové porty, na kterých pracuje replikační server. Údaj **„**Port**”** hned po názvu **„**Host_Name**”** určuje port, na kterém replikační server poskytuje replikační služby pro cluster DB. **„**Recovery_Port**”** určuje port poskytující služby pro obnovu databáze. **„**LifeCheck_Port**”** určuje port poslouchající odezvy o správném chodu od cluster DB.

Na straně cluster DB se nastaví porty pro stejnou funkci **„**Recovery_Port**”** a **„**LifeCheck_Port**”.** Dále se určí v **„**Rsync_Path**”** přístupová cesta k programu rsync používanému k replikaci nebo obnově databáze a typ používaného autentizačního mechanismu v **„**ssh**”**.

Existují 2 možnosti:

- „ssh -1" pro ssh verzi 1
- „ssh" pro verzi 2

Můžeme určit, zda-li užívat kompresi dat, v kolonce „Rsync_Compress", čas do vypršení při nečinnosti v „Rsync_Timeout", a v případě poruchy obnovy dat velkého objemu určit maximální přenosovou rychlost v „Rsync_Bwlimit".

V kolonkách „Pg_Dump_Path" a „Ping_Path" se určují přístupové cesty k programu Pg_dump a ping, Pg_dump je určen k zalohování databáze pomocí zápisu všech předchozích SQL příkazů[7].

Pomocí „When_Stand_Alone" popisujeme, jak se bude chovat cluster DB v případě výpadku všech replikačních serverů a cluster bude v režimu „Stand-Alone", tedy databáze v klasickém režimu. Můžeme použít dva způsoby:

- read_only - Z databáze se dá jen číst.
- read_write - Z databáze se dá číst i do ní zapisovat.

Následují určení vypršení času v případě žádosti o replikaci „Replication_Timeout", poté v případě odezvy o stavu „LifeCheck_Timeout" a interval posílání informací o stavu na replikační server „LifeCheck_Interval".

Nakonec je možné pomocí „Status_Log_File" a „Error_Log_File" určit soubory, do kterých se budou zapisovat průběžně stavy replikačních serverů a chybné zprávy od cluster DB.

Je také třeba nastavit v postgresql.conf, s kterými adresami bude postgresql komunikovat a na jakém portu. Také je nutné určit cestu k unixovému socketu.

```
#/usr/local/pgsql/data/postgresql.conf
listen_addresses = '*'          #* znamená poslouchat všechny adresy
port = 5432
unix_socket_directory = 'var/run/postgresql'
```

Musel jsem určit **„**unix_socket_directory**”,** poněvadž tato cesta není obecně určená.

Dále je nutné přidat v pg_hba.conf typ autentikace pro jednotlivé adresy.

# TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
host	all	all	195.113.113.152/27	trust
host	all	all	195.113.113.153/27	trust
host	all	all	195.113.113.154/27	trust

Pro adresy 152-154 jsem pro všechny databáze a uživatele povolil přístup do místní databáze.
Tuto celou konfiguraci je nutné provést stejně i pro druhou cluster DB.
Dalším krokem je konfigurace replikačního serveru.

```
#/usr/local/pgsql/etc/pgreplicate.conf
# informace o jednotlivých cluster DB
<Cluster_Server_Info>
        <Host_Name>         HA1         </Host_Name>
        <Port>              5432        </Port>
        <Recovery_Port>     7001        </Recovery_Port>
        <LifeCheck_Port>    7201        </LifeCheck_Port>
</Cluster_Server_Info>
<Cluster_Server_Info>
        <Host_Name>         HA2         </Host_Name>
        <Port>              5432        </Port>
        <Recovery_Port>     7001        </Recovery_Port>
        <LifeCheck_Port>    7201        </LifeCheck_Port>
</Cluster_Server_Info>
```

```
# informace o druhém replikačním serveru
<Replicate_Server_Info>
        <Host_Name>          HA2          </Host_Name>
        <Port>               8002         </Port>
        <Recovery_Port>      8102         </Recovery_Port>
        <LifeCheck_Port>     8202         </LifeCheck_Port>
</Replicate_Server_Info>
# vlastní konfigurace replikačního serveru
<Host_Name>                  HA1          </Host_Name>
<Replication_Port>           8001         </Replication_Port>
<Recovery_Port>              8101         </Recovery_Port>
<LifeCheck_Port>             8201         </LifeCheck_Port>
<RLOG_Port>                  8301         </RLOG_Port>
<Response_Mode>              normal       </Response_Mode>
<Use_Replication_Log>        yes           </Use_Replication_Log>
<Replication_Timeout>        1min         </Replication_Timeout>
<LifeCheck_Timeout>          3s           </LifeCheck_Timeout>
<LifeCheck_Interval>         15s           </LifeCheck_Interval>
```

Konfigurace je podobná jako konfigurace cluster DB, liší se v nastavení **„**RLOG_Port**”**, což je port, na kterém naslouchá příjem replikačních záznamů z druhého replikačního serveru, s čímž souvisí nastavení **„**Use_Replication_Log**”,** s hodnotami ano nebo ne. Další změnou je kolonka **„**Response_Mode**”** s možnými hodnotami normal nebo reliable, vysvětlenými v teorii o PGCluster.[6]

Problém, který může nastat je nerozeznání stanice podle hostname, proto jsem zařadil do souboru hosts následující řádky.

```
# /etc/hosts
127.0.0.1       localhost
195.113.113.152 HA1
195.113.113.153 HA2
195.113.113.154 HA1
195.113.113.154 HA2
```

4.2.3 Konfigurace SSH

Pro úspěšné spojení mezi cluster DB a replikačními servery je nutné nastavit správně ssh server, ssh klienta a vygenerovat klíče pro autentizaci stanic mezi sebou. Uvedu zde nejdůležitější nastavení pro pozdější výměnu rsa klíčů v konfiguračním souboru sshd_config pro ssh server. Celkové nastavení sshd_config je možné najít v příloze č. 1.

Nastavení sshd_config:

```
Protocol 2
HostKey /etc/ssh/ssh_host_rsa_key
PubkeyAuthentication yes
PermitEmptyPasswords no
```

Uvedeme zde, že chceme používat pouze protokol verze 2, umístění tajného klíče, povolení použití autentikace pomocí veřejných kličů a zabránění použití prázdných hesel.[8]

Přihlášení jako uživatelé postgres vygenerujeme 2 klíče, veřejný id_rsa.pub a tajný id_rsa. Tyto 2 soubory by se měly objevit ve složce /var/lib/postgresql/.ssh/. Poté nakopírujeme veřejný klíč na druhou stanici např. pomocí programu rsync do stejné složky a přejmenujeme na authorized_keys. Veřejný klíč musíme přejmenovat i na první stanici na authorized_keys. Poté vytvoříme klíče i na druhé stanici a provedeme stejnou operaci.

```
$ ssh-keygen -t rsa
$ rsync -ave ssh ./ssh/id_rsa.pub postgres@HA2:/var/lib/postgresql/.ssh/authorized_keys
```

Měli by jsme mít poté na každé stanici v souboru authorized_keys dva veřejné klíče, pomocí kterých se poté budou moci mezi sebou autentizovat. V našem případě, kdy je replikační server i cluster DB na stejné stanici, je proto nutné mít i vlastní veřejný klíč pro spojení mezi sebou.

4.2.4 Spouštění replikačního serveru a cluster DB

Nejprve by se měl spustit replikační server uživatelem postgres příkazem s volbou -D a za ní umístění konfiguračního souboru pgreplicate.conf:

```
$ /usr/local/pgsql/bin/pgreplicate -D /usr/local/pgsql/etc
```

Poté můžeme spustit samotnou cluster DB příkazem s volbou -D znamenající umístění databázového adresáře:

```
/usr/local/pgsql/bin/pg_ctl start -D /usr/local/pgsql/data
```

Po spuštění obou stanic je však nutné zařídit automatické spouštění replikačního serveru a obnovu cluster DB při startu. To jsem zařídil napsáním spouštěcích příkazů v souboru rc.local umístěném v adresáři /etc.

```
#/etc/rc.local
su -c '/usr/local/pgsql/bin/pgreplicate -D /usr/local/pgsql/etc' postgres
sleep 10
su -c '/usr/local/pgsql/bin/pg_ctl start -D /usr/local/pgsql/data -o "-R"' postgres
exit 0
```

Příkazem su -c 'příkaz' postgres spouštím příkaz jako uživatel postgres. Poté dám nutnou pauzu, např. 10 s pro spuštění replikačního serveru a poté spustím stejně cluster DB s volbou "-R" znamenající obnovu databáze z replikačního serveru.

4.3 Propojení Asterisk – PostgreSQL

Pokud chceme zajistit, aby se data o registracích z jedné ústředny replikovaly na druhou ústřednu, musíme použít další stupně v architektuře systému.

Asterisk sám o sobě používá pro ukládání dat svou vlastní databázi, která je určena jen pro potřeby asterisku a nedá se k ní konvenčně přistupovat. Asterisk má však možnost spolupracovat s tzv. ODBC (Open DataBase Connectivity), což je nástroj schopný propojit dohromady různé databáze. Přes tento konektor se pak může asterisk propojit již s databází PostgreSQL.

4.3.1 Instalace

Na obě stanice jsem nainstaloval ODBC, což se skládá z více balíčků, zejména knihoven:

- odbc-postgresql pro propojení odbc s postgreSQL
- unixodbc
- unixodbc-dev

poté je třeba znovu zkompilovat a nainstalovat asterisk a doinstalovat nejlépe přídavky asterisk-addons, jelikož musí zjistit, že jsou dostupné knihovny unixodbc, zároveň je nutné vybrat ve výběrovém menu menuselect moduly potřebné pro komunikaci s odbc:

- func_odbc
- func_realtime
- pbx_realtime
- res_config_odbc
- res_odbc

4.3.2 Konfigurace

Pro asterisk jsem v PostgreSQL vytvořil uživatele se jménem asterisk heslem welcome a databázi s názvem asterisk-db, jejíž majitelem je uživatel asterisk.

Co se týče PostgreSQL je třeba povolit komunikaci přes localhost v konfiguračním souboru postgresql.conf, což už bylo uděláno při konfiguraci PGCluster.

```
listen_adresses = '*'
port = 5432
max_connections = 100
```

Dále je nutné přidat pravidlo do souboru pg_hba.conf, aby se mohl uživatel asterisk přihlásit k databázi z adresy localhost s pomocí hesla.

#TYPE	DATABASE	USER	CIDR-ADRESS	METHOD
host	all	asterisk	127.0.0.1/32	md5

Zde jsem přiřadil uživateli asterisk přístup do všech databází s podmínkou zadání hesla s metodou šifrování md5.

Nyní můžu pokračovat konfigurací odbc konektoru. V souboru odbcinst.ini je třeba popsat přístup k ovladači databáze PostgreSQL a jeho nastavení.

```
[PostgreSQL]
Description     = ODBC for PostgreSQL
Driver          = /usr/lib/odbc/psqlodbca.so
Setup           = /usr/lib/odbc/odbcpsqlS.so
```

Vytvořil jsem v souboru odbc.ini relaci asterisk-odbc, na kterou potom budu odkazovat asterisk, je zde třeba uvést základní údaje, jak se bude asterisk přes odbc připojovat k databázi PostgreSQL. Běžné nastavení vypadá následovně.

```
[asterisk-odbc]
Description             = PostgreSQL connection to ‘asterisk’ database
Driver                  = PostgreSQL
Trace                   = No
TraceFile               = /tmp/psqlodbc.log
Database                = asterisk-db
Servername              = localhost
UserName                = asterisk
Password                = welcome
Port                    = 5432
ReadOnly                = No
RowVersioning           = No
ShowSystemTables        = No
ShowOidColumn           = No
FakeOidIndex            = No
ConnSettings            =
```

Upravím v asterisk konfiguračním souboru res_odbc.conf informace o odbc konektoru pro asterisk.

```
[asterisk-db]                       ; jméno databáze
enabled => yes
dsn             => asterisk-odbc    ; database source name
username        => asterisk
password        => welcome
pooling         => no
pre-connect     => yes              : Připojit před zavedením samotného asterisku
```

Nakonec v souboru modules.conf určujícím, které moduly asterisk načítá při startu je třeba zajistit načtení modulu res_config_odbc.so.

```
load => res_config_odbc.so
```

V souboru func_odbc.conf je třeba zaměnit příkazy read a write za readsql a writesql. Po restartu asterisku by jsme měli mít spojení asterisk – PostgreSQL, můžeme tedy vybrat, které údaje se budou načítat a číst přes PostgreSQL. Tyto informace se zapisují do souboru extconfig.conf.

```
[settings]
sippeers        => odbc,asterisk-db,sipcnf
sipusers        => odbc,asterisk-db,sipcnf
```

Informace, kde hledat tabulku se skládá ze jména ovladače, názvu databáze a názvu příslušné tabulky.[9]

4.3.3 Vytvoření tabulky pro funkci registrace

V PostgreSQL je tedy třeba v databázi asterisk-db vytvořit tabulku sipcnf tak, aby si v ní mohl asterisk ukládat informace o uživatelích. Nejprve vytvořím uživatele asterisk a databázi asterisk-db, přihlášený jako uživatel postgres.

```
$ createuser -P asterisk
```

Volba -P znamená vytvoření uživatele s heslem, poté odpovíme na otázku, zda-li bude superuživatelem, odpovíme ne. Na otázku, zda-li může tvořit databáze ano a nakonec, jestli může vytvářet nové uživatele odpovíme ne.

Poté vytvořím databázi asterisk-db.

```
$ createdb --owner=asterisk asterisk-db
```

Tabulku jsem poté vytvořil pomocí SQL příkazů takto.

```
$ psql asterisk-db
```

```
CREATE TABLE sipcnf (
id serial NOT NULL,
name varchar(80) NOT NULL,
host varchar(31) DEFAULT ‘dynamic’,
secret varchar(80),
ipaddr varchar(15),
port varchar(5),
regseconds integer DEFAULT 0,
username varchar(80),
context varchar(80) DEFAULT ‘phones’,
type varchar(80) DEFAULT ‘friend’,
defaultuser varchar(100),
fullcontact varchar(100),
regserver varchar(100),
useragent varchar(100),
lastms varchar(80),
CONSTRAINT sip_conf_pkey PRIMARY KEY (id)
)
WITH (OIDS = FALSE);
CREATE UNIQUE INDEX ‘name’ ON sipcnf USING BTREE (name);
GRANT ALL ON sipncf TO PUBLIC;
```

V prvním sloupci je identifikátor s typem proměnné serial, což znamená, že při zadávání nových údajů se bude toto číslo postupně inkrementovat od 1. Dále se pak vytvoří jednotlivé sloupce, do kterých bude asterisk server zapisovat údaje o registracích.

Pomocí funkce CONSTRAINT a PRIMARY KEY zajistíme jednoznačnost údajů ve sloupci id aby se nevyskytovaly řádky v tabulce se stejným identifikačním číslem. CREATE UNIQUE INDEX vytvoří jmenný index všech uživatelů.[9,10]

Nyní vložíme do tabulky pomocí SQL příkazu uživatele SIP s telefonním číslem 1200.

```
INSERT INTO sipcnf (name)
VALUES (1200);
```

Kromě údaje name se však vloží další údaje nastavené implicitně, pokud je nevložíme ručně, jedná se o údaje host, context a type.

Pokud se nyní zkusíme zaregistrovat s telefonním číslem 1201, asterisk se spojí s databází asterisk-db a najde v tabulce sipcnf jméno 1201 s údaji context = phones a type = friend. Telefon tedy zaregistruje a vloží další informace ipaddr, port, regseconds, fullcontact, useragent a lastms. Tyto informace asterisk ovšem v průběhu času postupně mění, zejména počet sekund do vypršení registrace (Tabulka č.2).

id	name	host	secret	ipaddr	port	regseconds	username	context	type	defaultuser
1	1200	dynamic		195.113.113.155	5071	1272460138		phones	friend	1200
fullcontact						regserver		user agent		lastms
sip:1201@195.113.113.155:5071;transport=udp								Ekiga/2.0.12		0

Tabulka č. 2: Tabulka sipcnf se zaregistrovaným číslem 1200

V Asterisk konfiguračním souboru extensions.conf jsem vytvořil jednoduchý dialplan pro 3 čísla 1200, 1201 a 1202:

```
; /etc/asterisk/extensions.conf

[phones]


#exten => 12,1,Answer()
#exten => 12,n,Background(enter-ext-of-person)
#exten => 12,n,WaitExten(30)


exten => 00,1,Dial(SIP/1200,10)
```

```
exten => 00,n,Playback(vm-nobodyavail)
exten => 00,n,Hangup()

exten => 01,1,Dial(SIP/1201,10)
exten => 01,n,Playback(vm-nobodyavail)
exten => 01,n,Hangup()

exten => 02,1,Dial(SIP/1202,10)
exten => 02,n,Playback(vm-nobodyavail)
exten => 02,n,Hangup()

exten => i,1,Playback(pbx-invalid)
exten => i,n,Goto(phones,12,1)
```

5 Zhodnocení dosažených výsledků

Chování celého systému budu analyzovat z třetí stanice umístěné v síti, kde budu pomocí síťového analyzátoru Wireshark zachytávat pakety síťového provozu.

5.1 Zhodnocení funkce přechodu IP adresy

Na obrázku 9 je vidět celkový výpis komunikace při přechodu virtuální IP adresy ze stanice HA1 na HA2.

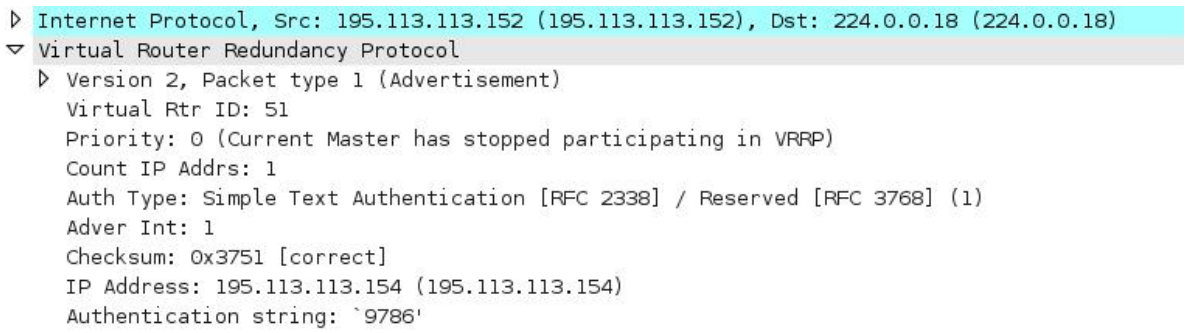
No.	Time	Source	Destination	Protocol	Info
21	20.061269	195.113.113.152	224.0.0.18	VRRP	Announcement (v2)
22	21.065780	195.113.113.152	224.0.0.18	VRRP	Announcement (v2)
23	21.359663	195.113.113.152	224.0.0.18	VRRP	Announcement (v2)
24	21.970692	195.113.113.153	224.0.0.18	VRRP	Announcement (v2)
25	22.976933	Vmware_5d:a2:f6	Broadcast	ARP	Gratuitous ARP for 195.1
26	22.976979	Vmware_5d:a2:f6	Broadcast	ARP	Gratuitous ARP for 195.1
27	22.976986	Vmware_5d:a2:f6	Broadcast	ARP	Gratuitous ARP for 195.1
28	22.976990	Vmware_5d:a2:f6	Broadcast	ARP	Gratuitous ARP for 195.1
29	22.976994	Vmware_5d:a2:f6	Broadcast	ARP	Gratuitous ARP for 195.1
30	22.976998	195.113.113.153	224.0.0.18	VRRP	Announcement (v2)
31	23.978093	195.113.113.153	224.0.0.18	VRRP	Announcement (v2)

Obr. č. 9: Převzetí IP adresy

Na začátku jsou spuštěny obě stanice, při vypnutí masteru HA1 (195.113.113.152), který před vypnutím vysílal v intervalu 1 sekundy VRRP pakety (obr. 10) na VRRP multicast (224.0.0.18), pošle jako poslední VRRP paket oznamující ukončení provozu nastavením priority na 0.(Obr. 11).

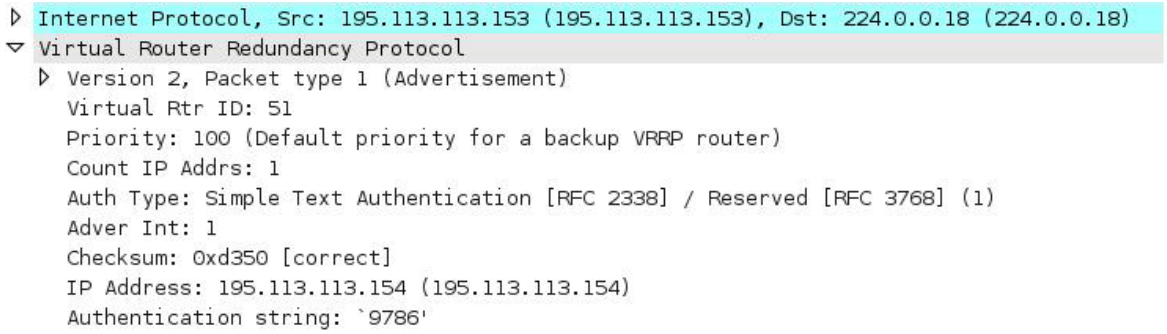
```
▷ Ethernet II, Src: Vmware_43:b2:a9 (00:0c:29:43:b2:a9), Dst: IPv4mcast_00:00:12 (01:00:5e:00:00:12
▷ Internet Protocol, Src: 195.113.113.152 (195.113.113.152), Dst: 224.0.0.18 (224.0.0.18)
▽ Virtual Router Redundancy Protocol
  ▷ Version 2, Packet type 1 (Advertisement)
    Virtual Rtr ID: 51
    Priority: 200 (Non-default backup priority)
    Count IP Addrs: 1
    Auth Type: Simple Text Authentication [RFC 2338] / Reserved [RFC 3768] (1)
    Adver Int: 1
    Checksum: 0x6f50 [correct]
    IP Address: 195.113.113.154 (195.113.113.154)
    Authentication string: `9786'
```

Obr. č. 10: Výpis VRRP paketu s prioritou 200



Obr. č. 11: VRRP paket při ukončování činnosti

Poté co stanice HA2 vyhodnotí změnu priority od HA1, přepne se do stavu master a začne sama posílat VRRP pakety s prioritou 100 (Obr. 12).



Obr. č. 12: První VRRP paket od HA2

V následující komunikaci byly posílány rámce ARP gratuitous (Obr. č. 13), které oznamují změnu IP adresy vzhledem k MAC adrese. Ty byly posílány na MAC broadcast adresu (FF:FF:FF:FF:FF:FF).

```
▷ Ethernet II, Src: Vmware_5d:a2:f6 (00:0c:29:5d:a2:f6), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▽ Address Resolution Protocol (request/gratuitous ARP)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: Vmware_5d:a2:f6 (00:0c:29:5d:a2:f6)
    Sender IP address: 195.113.113.154 (195.113.113.154)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 195.113.113.154 (195.113.113.154)
```

Obr. č. 13: Rámec ARP gratuitous

Po úspěšném spuštění stanice HA1 se opět virtuální IP adresa přehodí na HA1, jelikož má vyšší prioritu (200).

5.2 Zhodnocení funkce replikace databází

Obr. č. 14: SSH komunikace mezi HA2 a HA1

Na obrázku č. 14 je vidět situace při spouštění stanice HA1, kdy začíná obnova databáze z replikačního serveru na stanici HA2 pomocí zašifrovaných zpráv. Nejprve si ssh server a klient domluví výměnu klíču a poté začíná samotná výměna zašifrovaných zpráv.

5.3 Vlastnosti registrací a hovorů

Díky uložených identických registrací na obou stanicích je zachována registrace telefonu I při výpadku master stanice a telefon je zaregistrován na záložní stanici. Pokud je však před výpadkem uskutečněn hovor, je tento hovor po přechodu IP adresy na záložní stanici ukončen, jelikož údaje o hovorech jsou nepřenositelné (Obr. č. 15). V našem případě vyšle při klasickém vypnutí stanice SIP server zprávu BYE.

Při přechodu IP adresy zpět na master stanici je však volání nepřerušeno, protože telefon komunikuje se SIP serverem s jeho reálnou IP adresou.

1343	13.218638	195.113.113.152	224.0.0.18	VRRP	Announcement (v2)
1492	13.723710	195.113.113.152	94.112.44.68	SIP/SDP	Request: INVITE sip:1200(
1493	13.724310	195.113.113.152	94.112.44.68	SIP	Request: BYE sip:1200@94
1501	13.753266	94.112.44.68	195.113.113.152	RTCP	Sender Report Source d
1507	13.833641	195.113.113.153	224.0.0.18	VRRP	Announcement (v2)
1526	13.859270	94.112.44.68	195.113.113.152	SIP	Status: 200 OK
1527	13.859284	94.112.44.68	195.113.113.152	SIP/SDP	Status: 200 OK, with ses
1528	13.859912	195.113.113.152	94.112.44.68	ICMP	Destination unreachable
1529	13.860093	195.113.113.152	94.112.44.68	ICMP	Destination unreachable

Obr. č. 15: Ukončení hovoru při přechodu IP adresy

Další nevýhodou je nemožnost nové registrace v průběhu obnovy jedné z cluster DB, je to zapřičiněno zamknutím databáze během obnovy, kdy nejde zapisovat ani číst. Tato obnova trvá asi 10 sekund, bude se však lišit v závislosti na velikosti databáze.

6 Závěr

Existuje celá řada různých řešení, které se zabývají tématem vysoké dostupnosti, již méně je však vhodných pro vysoce dostupný asterisk SIP server, který má jisté specifika provozu.

K identifikaci funkčního serveru podle IP adresy jsem vybral Keepalived program, který zajišťuje přenositelnost virtuální IP adresy mezi stanicemi v závislosti na jejich funkčnosti. Použil jsem nejvhodnější řešení a to PGCluster synchronní multi-master replikaci databází obsahující PostgreSQL databázi pro přenos dat používaných asteriskem oběma směry. Nakonec jsem asterisk propojil s databází PostgreSQL jediným nabízeným řešením, a to tzv. ODBC konektorem.

Po celkovém navržení řešení a jeho implementací na stanice se mi podařilo zprovoznit vysoce dostupný SIP server, jenž je provozuschopný, pokud je vždy aspoň jedna ze dvou stanic funkční. Vzhledem k malé pravděpodobnosti výpadku obou stanic najednou, což může být způsobeno výpadkem elektřiny je toto dostačující. Výpadek elektřiny se může řešit např. záložními zdroji, či podobně.

Mezi nevýhody u tohoto řešení je rozpad spojení v případě výpadku stanice, na které je zrovna veden hovor a nemožnost zaregistrování na SIP server v průběhu obnovy jedné z databází.

Literatura

1. MORIC, Lubomír - VILČ, Jaroslav. *VRRP*. Ostrava: VŠB-TUO. FEI, 2006 [online], URL: <http://www.cs.vsb.cz/grygarek/SPS/projekty0506/vrrp.pdf>

2. FILKA, Michal. *Vysoce dostupný firewall s Keepalived* [online], URL: < http://www.root.cz/články/vysoce-dostupny-firewall-s-keepalived/>

3. CASSEN, Allexandre. *Keepalived User Guide* [online] URL: < http://keepalived.org/pdf/UserGuide.pdf>

4. *High Availability Linux project* [online], URL: < http://www.linux-ha.org >

5. *DRBD* [online], URL: < http://www.drbd.org/ >

6. The PostgreSQL Global Development Group. *PGCluster* [online], URL: < http://pgcluster.projects.postgresql.org/index.html >

7. *PostgreSQL 8.3.10 Documentation* [online], URL:<http://www.postgresql.org/files/documentation/pdf/8.3/postgresql-8.3-A4.pdf>

8. *Manual sshd_config* [online], URL: < http://unixhelp.ed.ac.uk/CGI/man-cgi?sshd_config+5 >

9. MEGGELEN, Jim Van - MADSEN, Leif - SMITH, Jared. *Asterisk - the future of telephony v2*, O'Reilly [online], URL: < http://cdn.oreilly.com/books/9780596510480.pdf >

10. *Asterisk realtime PostgreSQL* [online], URL: < http://www.voip-info.org/wiki/view/Asterisk+RealTime+PostgreSQL>

Seznam Příloh

Přílohy na CD

Text Diplomové práce

Programy

Asterisk-1.6.1.18

Asterisk-addons-1.6.1.2

pgcluster-1.9.0rc5

Konfigurační soubory

PGCluster

cluster.conf

pg_hba.conf

pgreplicate.conf

Keepalived

keepalived.conf

Asterisk

modules.conf

res_odbc.conf

extensions.conf

extconfig.conf

ODBC

odbc.ini

odbcinst.ini

SSH

- ssh_config
- sshd_config

rc.local

hosts